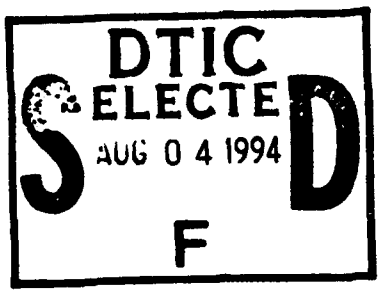
Active Nodal Task Seeking for High-Performance, Ultra-Dependable Computing

Jien-Chung Lo, Donald W. Tufts and James W. Cooley
Department of Electrical and Computer Engineering
The University of Rhode Island
Kingston, RI 02881-0805



 This paper is scheduled for publication in an upcoming issue of *IEEE Transactions on Aerospace & Electronic Systems* (tentative schedule: Volume 31, Issue Number 4, October 1995).

Active Nodal Task Seeking for High-Performance, Ultra-Dependable Computing

Jien-Chung Lo, Donald W. Tufts and James W. Cooley
Department of Electrical and Computer Engineering
The University of Rhode Island
Kingston, RI 02881-0805

Abstract

The Active Nodal Task Seeking (ANTS) approach to the design of multicomputer systems is named for its basic component: an Active Nodal Task-Seeker (ANT). In this system, there is no load balancing or load sharing, instead, each ANT computing node is actively finding out how it can contribute to the execution of the needed tasks. A run-time partition is established such that some of the ANT computing nodes are under exhaustive diagnosis at any given time. An ANTS multicomputer system can achieve a mean time to failure of more than 20 years with just 8 computing nodes and 3 buses, while the minimum requirements are 3 computing nodes and 1 bus, and with a worst case computing node failure rate of 5×10^{-4} per hour.

This work has been motivated by the need to develop high-performance multicomputer systems for radar, active and passive sonar, and electronic warfare that can provide ultra-dependable performance for more than 20 years without field repairs. We argue that high performance is also an attribute of an ANTS computing system, because the overhead of dynamic task scheduling is reduced and because efficient use is made of the available processing resources.

[1]This work was supported in part by the National Science Foundation under grant MIP-9308085 and by the Office of Naval Research under grant N00014-94-1-0479

1 Introduction

Active Nodal Task Seeking (ANTS) is an approach to the design of high-performance, ultra-dependable multicomputer systems. The goal is to design an ultra-dependable, real-time, and distributed system for computationally intensive signal and data processing applications. The standard for ultra-dependability is set at a mean time to failure (MTTF) exceeding 20 years. The hardware organization of ANTS is a distributed systems with multiple-bus connected stand-alone computing nodes. This system uses a novel operating mode to achieve the ultra-dependability and maintain a high degree of computational performance.

The only direct predecessor to ANTS is the now-forgotten, but very successful, Safeguard multiprocessor system of Bell Telephone Laboratories. Before reading on or looking at our list of references, we urge the reader to search his or her memory for information about the existence or architecture of the Safeguard system. In books on the design of computer systems [1, 2, 3] no references to or mention of Safeguard appears. Safeguard [4] was the high-performance, dependable multiprocessor for computation and control of the Safeguard antiballistic missile (ABM) defense system of the early 1970's. ANTS extends and generalizes the approach of Safeguard, and we intend to provide performance and dependability analysis for ANTS multicomputer systems. Two other predecessor systems are FTMP [7] and SIFT [8]. Unlike ANTS, these systems use at least three times the resources required by the application. Triads of processors are assigned to execute task segments. Also FTMP adopted a fully synchronous approach which uses a hardware implemented bit-by-bit voting on all transactions. The asynchronous nature of ANTS is closer to that of SIFT.

The term dependability collectively describes the common fault-tolerant system measurements, such as reliability, availability, MTTF, *etc.*. Depending on the mission, one or more of these measurements are used in the system specification [5]. For instance, electronic switching systems (ESSs) are designed to achieve high availability [6]. Avionics control systems, such as FTMP and SIFT are designed to achieve ultra-high reliability for a short mission time. For military and space-bound applications, a long MTTF is required to ensure the operation under severe circumstances [5].

Distributed systems are potentially fault-tolerant. However, utilizing this intrinsic capability for fault-tolerance is not a straightforward task. Many existing dependable systems

still use special hardware designs to achieve the desired level of dependability. The Advanced Architecture On-board Processor by Harris Corporation [9, 10] uses self-checking RISC processors and chordal ring. The Advanced Automation System by IBM [11, 12] needs redundant components at each subsystem due to its wide area distribution nature. The on-board computer of the Japanese satellite Hiten [13] uses stepwise negotiating voting, which is a combination of mutual checking by data comparison and self-checking.

Special hardware components may not be as necessary if an appropriate software approach is applied. In Delta-4 [14], active replication of software programs is used to ensure fault-tolerance. A similar strategy is used in Manetho [15], where each application process is replicated by a set named: troupe. These systems are distributed systems with no special hardware designed for fault-tolerance. Unfortunately, replication of programs also means a significant reduction in system computational performance.

Existing dependable multicomputer or multiprocessor designs assume that the computing nodes or the processors are passive in their operating mode. Load balancing or load sharing [16] is thus required because the idle computing nodes remain idle until new jobs are distributed to them. The busy nodes are responsible for distributing tasks to the idle ones.

From the fault tolerance perspective, active nodal task seeking (ANTS) offers several advantages:

- On-line error checking is simple and straightforward. The two extreme failure modes: *fail-silent* and *fail-uncontrolled* [14] are easily detected since the failed computing node will not be actively seeking a new task in a fixed time frame.

- Fault tolerance can be attained without seriously degrading the system performance. There is no need for special hardware or replication of programs in ANTS and thus no serious performance degradation. If ultra-reliability is required, during the critical phase of the mission, jobs are triplicated in an asynchronous, distributed mode. In this case, the voting function is implemented in software similar to that in SIFT [8].

- There is no need for synchronization of nodes or programs. The active nodal task-seeker (ANT) computing nodes operate asynchronously and independently. Of course, they must work cooperatively. And each ANT node, when seeking a new task, must be aware of the pertinent work that has been done by itself and other ANT nodes as it is

posted in the bulletin board (task tables). Each ANT node keeps its own copy of the task table. The entries of these distributed task tables are updated via broadcasting of task acquiring and task completion messages over the communication network.

- The ANTS system is designed such that no single dependability-critical component exists in the system. Error checking functions are handled distributively. Near coincident failures or even multiple failures can only degrade the system performance temporarily but cannot paralyze the continuous operation of an ANTS system.

In this paper, we will only briefly compare the ANTS concept with conventional approaches for execution of real-time tasks in a distributed computing environment. This is an area of current concentration in our research. The ANTS concept is efficient since only little time and little computational resources are used for scheduling tasks on the computing nodes. Therefore, computing nodes can do more work on the real-time applications in any fixed time interval. Of course, this requires an extra, one-time effort when developing and programming the real-time tasks. The ANTS concept is effective because the system throughput is close to that of an ideal system in which each processor that is not under repair knows exactly how to contribute to the current set of real-time application tasks or testing tasks. Performance analysis of ANTS will be presented in a separate paper.

This paper addresses the design features of ANTS which are aimed at enhancing the dependability. An analytical model is developed to evaluate the mean time to failure (MTTF) of the ANTS system. The design goal is set at achieving an MTTF of 20 years and more. We show that a simple combination of 8 computing nodes and 3 buses, with minimum requirement of 3 computing nodes and one bus, could achieve this goal, assuming that the computing node failure rate is 5×10^{-4} per hour, or a MTTF of 2,000 hours. This means that commercially available components are suitable for constructing an ultra-dependable ANTS system. Also the specialized applications for which ANTS is intended, such as a digital-cellular-radio based station, a central-office digital-communications signal processing interface, an active/passive sonar system, or a radar system naturally lend themselves to efficient and effective reasonableness checks. Error detection/correction codes provide such checks in communication systems. And the physical consistency of estimated parameter values provide such checks in sonar and radar.

2 ANTS Multicomputer System

ANTS is based on the concept that each active nodal task-seeker (ANT) node must be capable of stand-alone operation in which it accesses a bulletin board to find the state of high priority tasks and post its intended contribution during a prescribed time slice of work. In this paper, we use a simple architecture to demonstrate the features and capability of ANTS concept so that we can concentrate on analyzing dependabilities. This architecture will eventually be described in more detail as modifications are incorporated to provide improved performance. Motivation will be derived from the requirements of digital signal processing algorithm implementation.

Figure 1 shows a hardware organization of ANTS: stand-alone computing nodes interconnected by buses.

2.1 Run Time Partitioning

The computing nodes, as well as the buses, are partitioned into three groups during run time: the up-and-running green group, the stop-and-checking yellow group and the red group for removed faulty nodes and buses. The number of computing nodes allocated for the green group must always be greater than the minimum requirement for execution of the applications at any given time. The rest of the computing nodes are then in the yellow group unless they have been found to fail, in which case they are removed to the stopped, red group. The grouping is based on the status of the computing nodes and/or buses. Therefore, there is no visible physical grouping, and no need for special hardware design for switching between the two run-time groups.

Since each computing node seeks work for itself, placing a computing node in the yellow, checking group is accomplished by making the diagnostic program the highest priority task for it, through a timer for example. To exercise the bus, one of the yellow computing nodes will gain exclusive access to one of the buses and check its vitality. In other words, there are two types of diagnostic programs: one checks computing node only and the other checks both a computing node and a bus.

Besides the diagnostic and other bookkeeping functions, all jobs are initiated by input/output devices. We assume that all ANT nodes acknowledge the job initiations and

create the entries in their respective task table locally. There is a certain degree of conceptual similarity between ANTS and FTMP in the assignment of jobs to processing units although the implementation differs in many aspects. In FTMP, as soon as the next available triad is formed, it is assigned with the next waiting job. In ANTS, the next available node grabs the next available task. The main difference being that each ANTS node maintains its own task table, which is coherent with the task tables of other ANTS nodes.

2.2 Concurrent Error Detection for Control Transfer Errors

The failure modes of an ANT node can be classified into the following two categories: *control transfer errors* and *data manipulation errors*. A control transfer error may manifest itself in the following three scenarios:

1. Fail-silent: the faulty ANT node cannot send out any message over the network due to the failure.

2. Babbling: the faulty ANT node keeps sending meaningless messages over the network and may eventually block out all communication. This is a type of fail-uncontrolled failure.

3. Tampering: the faulty ANT node is failed in such a way that the electrical characteristics of the communication network is destroyed, *i.e.* a short to the ground on the communication wire. This is a worst case scenario with a relatively small probability and can only be handled by special hardware design.

Since each computing node has an active role in seeking out work, a fail-silent node is easy to identify. A fail-silent node will not ask for a job for an extensive period of time. If a failure occurs after a new job has been acquired, then the failed node is unable to report a completion, even if it is still possible, with the fixed time period. For easy real-time scheduling, each task is defined such that it can be completed in a fixed time frame. Therefore, we may monitor the completion time of each task to detect the fail-silent nodes.

For general purpose applications, partitioning of a program into fixed completion time frame tasks requires a special compiler that can accurately estimate the execution time. This will not be the case for the proposed system. First, the system is designed to handle sonar

and/or radar signal processing; the programs to be executed have been specified. Secondly, the fixed time frame constraint is not a hard constraint. The partitioning of a program must also take into account the parallelism between the tasks. In fact, a guaranteed parallelism between the tasks is the first priority in the task division. Therefore, some tasks may complete much earlier than the pre-selected time. A repeating process is required to select the proper time frame of the system.

The time limit set for the time-out checking must be greater than the time selected for the fixed time frame. The main reason for this consideration is that the communication time is a random variable. If we insist on declaring a node failure whenever it exceeds a tight time limit, we may have a great number of false alarms.

When a faulty ANT node sends out meaningless messages and jams the traffic on the communication network, it can be detected easily since all ANT nodes check all broadcasting messages on the network. Further, all broadcasting message has a predefined format and protocol. An incorrect transmission can be detected almost immediately. The electrical characteristics of the network may also be destroyed by a failed ANT node such that further communication is impossible. This type of failures can be avoided using similar design as the Bus Guard (BG) in FTMP [7]. However, we caution here that this type of design is device dependent. Detailed designs can only be derived for a specific implementation. We also note that this type of failure may not be a catastrophic one. When several independent peripheral devices are used to handled buses, respectively, we may see that the probability of all bus communications being wiped out is negligibly small.

In an ANTS system, the communication of an ANT node is monitored by two other designated ANT nodes. In other words, an ANT node is responsible for monitoring two other ANT nodes. For this purpose, each ANT node in the green group is assigned with a number. Each ANT node checks the broadcasting messages constantly not only for its own message but also for the messages directed to the two ANT nodes that it monitors. When a task is acquired by an ANT node and no further message about the completion of that task was sent by that ANT node for an extensive period of time, the two ANT nodes that monitor this ANT node will sense a time out. Since all ANTS operations are asynchronous, the two ANT nodes may not detect this time-out simultaneously. The ANT node with a higher number must take the initiative and sends inquiry to the other ANT node that monitors the

node in question. When a confirmation is received, the ANT node with time-out is declared faulty and is moved to the yellow group. A similar situation occurs when an ANT node sends some meaningless messages over the network.

Let's use a simple example to demonstrate this technique. Assuming that we have four nodes in the green group and they are numbered from one to four. Node 1 is monitored by node 2 and node 4, and Node 2 is monitored by node 1 and node 3, *etc.* If node 3 did not send out a message within the predefined time, and node 4 senses this situation, node 4 will first communicates with node 2 for verification. If node 2 is fault-free, it will confirm the finding of node 4 and declare that node 3 is failed.

If node 2 has failed and does not answer node 4 within the pre-defined time, node 4 will send an inquiry to node 1 to check the validity of node 2. If node 1 verifies that node 2 has not been responding for a long time, nodes 2 and 3 are both declared faulty. Note that when nodes 2 and 3 are both faulty, node 1 will not take the initiative to send out an inquiry since it has a lower number. In this case, node 1 is waiting for node 3 to send the inquiry.

We note that the above concurrent error detection techniques for control transfer errors are for ultra-long MTTF missions. To provide higher reliability, the executions of critical tasks can be triplicated to provide the on-line error masking capability.

2.3 Concurrent Error Detection for Data Manipulation Errors

Failures may affect only the data manipulations. Computed results may be incorrect. To detect these erroneous computed results, a reasonableness checking function is associated with each job. It checks all the computed results before an acceptance. Special conditions on some operations can also be derived for the checking purpose. For example, in a Fast Fourier Transform (FFT) computation, the sum of the squared absolute values of the inputs is equal to the sum of squared absolute values of the transformed outputs. Similar checking is also possible for some matrix operations. Other type of conditions such as the *a priori* knowledge of physical limitations and special mission environment parameters will also be used in the reasonableness checking. For instance, the speed of objects monitored by a radar will lie within a physical reasonable range.

Reasonableness checking cannot guarantee full fault coverage. In contrast, a triplicated

program can guarantee not only a full fault coverage, but also an immediate correction. Whereas, using the reasonableness checking, retry or recomputation may be required. However, the consideration here is that the such a fault coverage may not be necessary and that the triplication degrades the system performance significantly. High fault coverage is not necessary for data manipulation errors because data integrity is not always the criterion of dependability during the entire mission. In a typical radar tracking mission, unless a close contact with enemy has been engaged, we can tolerate a few ms of not 100% correct, but reasonable, radar outputs.

If ultra-reliability is required during a particular phase of the mission, software replication as in [14, 15] can be used. This can be easily implemented on ANTS. Three entries instead of one will be generated in the task tables for each task of the critical job. This critical job is considered completed when all replications have been completed and the checking result is successful. The voting function is implemented in software as a final task of the job, similar to that in SIFT [8]. Moreover, the concept of N-version programming [17], where the replicated programs are written by different software teams, can also be easily implemented on ANTS.

An appropriate application environment can be found in the space shuttle on-board computer system [18]. This system is designed for high availability with no fault masking capability for most of its mission time. However, during critical phases of the mission, *e.g.* take-off and re-entry, triplication with three different versions of programs are executed to ensure a high reliability in these short periods. We envision a similar application environment and thus the design philosophy is very different from that found in FTMP and SIFT.

2.4 Recovery and Re-Enlistment

When a potentially failed computing node is identified in the green group, it is placed in the yellow, checking group immediately. Of course, one of the nodes in the yellow group must be released to the green group to maintain the operational requirement. There are three different situations for different recovery strategies:

- The failed node is executing a testing function, not an any operational task, when it fails. There is no need for recovery at all.

- The failed node is executing an operational task when it fails. Every fault-free ANT node will re-initialize the entry of that task such that it will again be available for execution by a free ANT node seeking work. Because the running time of each task segment must be short in order to satisfy real-time constraints, very little computations have been lost.

- The failed component is a bus, and once it is identified, will be placed in the yellow group immediately. There is no need for recovery if the bus contains no memory elements.

These recovery strategies apply to only the ANT nodes in the green group. If a node in the yellow group is found faulty, a cold start procedure is initiated and then the diagnostic program is used to verify the checking result. When an ANT node is released from the yellow group to the green group, its task table is created from communicating with the ANT nodes that monitor it.

The system makes no distinction between a failed subsystem or a subsystem for routine check up in the yellow group. The only difference is that a failed subsystem will go through the cold start procedure before the diagnostic program checking. In addition, the failure type and the time of failure will be recorded in a failure log of the failed node after the cold start. If the result of diagnostic program checking confirms a failure, the node is removed from the system into a red stop group. If no failure could be identified, the node will be considered operational and will be released back to the green group. We call this re-enlistment. Re-listment occurs whenever the node failure is transient: a failure that appears only for a short time. Since a cold start is performed after a failure is identified, a transient failure will no longer exist in the subsystem.

Since intermittent and transient failures occur more frequently during run time [18], especially in modern VLSI-based systems [19], there is no need to remove a component once a failure has been recorded. The record kept at each node will indicate the number and frequency of failure history. If a computing node fails frequently, it will be removed from the system. This situation could be induced by two possibilities: the node is on the brink of a permanent failure thus increasing the frequency of transient failure occurrences; or there exists a failure which is undetectable by the diagnostic program. It is impossible to design a

diagnostic program with a 100% *realistic failure* coverage, although it may guarantee a 100% coverage on modeled failures.

The re-enlistment of an ANT node with transient or intermittent failure makes the ANTS system behave similar to a repairable system in dependability evaluation. The ANT nodes in the yellow group can be considered as hot spares. As for maintainability, the diagnostic programs which are incorporated in an ANTS design can be very useful for field engineers to perform corrective maintenance. During run time checking, the purpose of the diagnostic program is to verify or to identify the existence of failures. For field repair, the diagnostic program should be augmented with the capability to locate the failures.

3 Dependability Modeling

In the following analysis, we assume that the failure rates of all components are exponentially distributed. We also deal with only the single failure cases. The main reason is because near-coincident failures or even multiple failures in an ANTS system will not induce a total system failure. Although, the inclusion of coverage factors and the use of more accurate failure function, such as Weibull, will give a more accurate evaluation [5], the presented analysis is sufficient in providing information for the system design and fine tuning. For this analysis, the field repair by maintenance engineers is not considered.

The ANTS is considered as a serial system with a KP-out-of-NP, (NP, KP), computing subsystem and a KB-out-of-NB, (NB, KB), bus subsystem. The system failure rate is the sum of the failure rates of the two subsystems [20]. Therefore, the MTTF of the system may be computed as

$$MTTF_{Sys} = \frac{1}{\frac{1}{MTTF_P} + \frac{1}{MTTF_B}} \tag{1}$$

where $MTTF_P$ is the MTTF of (NP, KP) computing node subsystem and $MTTF_B$ is the MTTF of the (NB, KB) bus subsystem. The system is considered to have failed if less than KP processors or KB buses remain fault-free.

Even though there is no repair facility, the dependability of ANTS multicomputer system could be modeled using birth and death process due to the re-enlistment. Figure 2 shows the Markov model for the subsystems. The failure rate, λ will eventually be replaced by λ_p, the computing node failure rate, or λ_b the bus failure rate. The failures considered include both

transient and permanent types. Therefore, λ_p and λ_b represent the arrival rate of transient and/or permanent failures. The birth rate in Figure 2 is ρD, where ρ is the rate for diagnostic program checking and D is defined as

$$D = \text{Prob}\{\text{failure is detectable}\} \times \text{Prob}\{\text{failure is transient}\} . \tag{2}$$

In other words, D is the product of failure coverage of the diagnostic program and the ratio of transient failures.

The states in Figure 2 represent the number of fault-free processors or buses. The steady-state probabilities of states in Figure 2 are expressed as [20]:

$$p_k = (\frac{\lambda}{\rho D})p_{k+1} \tag{3}$$

for $0 \leq k < N$. Since

$$p_N = 1 - \sum_{i=0}^{N-1} p_i , \tag{4}$$

we find

$$p_N = \frac{1}{\sum_{i=0}^{N} (\frac{\lambda}{\rho D})^i \frac{N!}{i!}} . \tag{5}$$

For the steady-state availability A_{ss} of a K-out-of-N subsystem, K components must be operational in an N component initial set up. The formula for A_{ss} is

$$A_{ss} = \sum_{i=K}^{N} p_i . \tag{6}$$

The mean time to failure (MTTF) is the expected time of system survival. Thus, the steady-state availability is the fraction of time that the system is operational [21], such that

$$A_{ss} = \frac{MTTF}{MTTF + MTTR} . \tag{7}$$

Here, the MTTR is the mean time to repair of the system. It is the time for a repair after the system has failed (has less than K fault-free components). Since the repair mechanism in ANTS is a diagnostic step followed by a re-enlistment step,

$$MTTR = \rho D . \tag{8}$$

Combining the above two equations, we find

$$MTTF = \frac{\rho D A_{ss}}{1 - A_{ss}} . \tag{9}$$

This model is used to evaluate the MTTF of computing subsystem and that of bus subsystem: $MTTF_P$ and $MTTF_B$, respectively. The MTTF of the system is then computed using equation (1).

4 Mean Time To Failure

Let us consider a design goal of the ANTS multicomputer system achieving a MTTF of more than 20 years, or 175,200 hours. We use the model developed in the previous section to examine the effects of computing node failure rate and the D to the system MTTF. We show that the ANTS concept could indeed tolerate a wide range of computing node failure rate as well as D (product of diagnostic program failure coverage and ratio of transient failures).

4.1 Effects of Computing Node Failure Rates

Figures 3 and 4 show the system MTTF of various ANTS configurations and a conventional hybrid N-modular redundancy system (HNMR). The graphs are for (NP, KP)=(8, 3), (8, 4), (16, 10) and (16, 12), respectively. For the HNMR system, we consider a 3-out-of-8 system with 32 standby spares. We further assume that these 32 cold spares will not fail while standby. According to [20], the MTTF of a HNMR(N, M)/S system is derived as:

$$MTTF_{HNMR} = \frac{S}{N\lambda} + \sum_{i=M}^{N} \frac{1}{i\lambda} \tag{10}$$

where λ is the component failure rate.

Figure 3 assumes a bus failure rate of 10^{-5} per hour, while Figure 4 assumes that the bus failure rate is 10^{-4} per hour. In each case of ANTS configuration, decreases in MTTF numbers are observed when the bus failure rate is higher. However, the decrease in MTTF is insignificant compare to the 10 times increase in the bus failure rate. In the above results, we assume ρ=0.1, or an averaging diagnostic program checking time of 10 hours. This is indeed a worst case assumption. Further, D=0.8 is assumed.

Obviously ANTS yields a much higher MTTF than a conventional HNMR system. From Figure 4, an ANTS system with NP=16 and KP=12 could achieve more than 20 years of operations even when the computing node failure rate is 3×10^{-4} per hours. The above results

strongly imply that there is no need for ultra-dependable computing nodes. A relatively inexpensive implementation of ANTS to achieve the MTTF goal is feasible.

4.2 Effects of Failure Coverage and Transient Failure Ratio

One of the reasons that the ANTS concept is a significant improvement with respect to the conventional HNMR is the fact that the failed computing nodes in ANTS are re-enlisted after a successful diagnostic program check out. The dependability of a HNMR could be enhanced using the same technique, but the system computational performance of HNMR is not comparable to that of ANTS. For instance, the HNMR(32, 8)/32 system has a total of 40 computing nodes: eight active ones and 32 cold spares. The entire system is used as a uniprocessor system. Whereas an ANTS system is a true distributed computer system, all 40 computing nodes could be fully utilized to perform useful tasks.

According to [20], a high system availability could be achieved by one very efficient "repairman" rather than by an unlimited number of repairmen. In the above dependability model, we use ρD to model the ANTS "repair" process. In Figures 5 and 6, we plot the system MTTF against the values of D to determine the effect of low diagnostic program failure coverage and low transient failure ratio.

The plots in Figures 5 and 6 have identical parameters, except for the bus configurations. In Figure 5, the best case configuration is a (16, 8) ANTS system, which achieves the desired goal with D as low as 0.35. In the worst case, a (32, 28) ANTS system, the lowest possible D is about 0.75. Form Figure 6, we find that the requirement of D decreases with a (4, 2) bus configuration for (8, 4) and (32, 28) configurations. The (8, 3) and (16, 8) configurations need a higher D to achieve the same level of MTTF when using a (4, 2) bus configuration.

We assume here that ρ=0.1, or a 10 hours diagnostic program execution time. We emphasize that this is a worst case assumption. The actual diagnostic program should execute at a much faster rate.

There are two parameters involved in deriving D: diagnostic program failure coverage and transient failure ratio. The transient failure ratio is an environmental parameter, in which we have no control. The failure coverage is the percentage of failures that can be detected by the diagnostic program. A higher failure coverage is possible by a carefully designed

diagnostic program. We note that there is a trade-off between the failure coverage and the program execution rate. Intuitively, a diagnostic program with higher failure coverage takes longer time to complete.

Assuming that the failure coverage of the diagnostic program is 90%, to reach D=0.35 means that the transient failure ratio must be 0.39. For D=0.75, the transient failure rate must be 0.83. In other words, given a transient failure ratio, there exists an ANTS configuration that achieves the goal of MTTF exceeds 20 years.

The study on space shuttle computers [18] assumes that the "self-test" program, which is equivalent to the diagnostic program in ANTS, has a fault coverage of 96%. Also, a 2:1 and a 4:1 "transient-to-solid", *i.e.* transient-to-permanent type failures, are assumed in the dependability analysis. Using the terminology in this paper, a 2:1 ratio means the transient failure ratio of 0.66, and a 4:1 ratio means the ratio is 0.8.

5 Performance Implications

The goal of the ANTS concept is to achieve both high-performance and ultra-dependability. In this section, we briefly discuss the performance implications of an ANTS computing system.

In a real-time distributed computing environment, some computing nodes may be busy and may be over loaded, while some computing nodes in the system are idle. An uneven load may result in missing the deadlines of jobs. Obviously, a high level of utilization of computing nodes is an essential requirement for real-time applications.

A user job is first partitioned into a sequence of tasks, each with specified predecessor/successor relationships. Conventional approaches involve the following four phases [22]:

1. Task definition - specify the identity and characteristics of the tasks.

2. Task assignment - the initial placement of tasks on processors [22, 23, 24].

3. Task allocation (scheduling) - local scheduling of individual tasks to computing nodes with overall progress consideration [25, 26].

4. Task migration (load sharing) - dynamic reassignment of tasks to computing nodes in

respond to changing loads [27] or system reconfiguration [26]. Existing techniques are all source-initiated or server-initiated [27].

5.1 Properties of ANTS Concept

Much of the above work does not need to be done at run time if the Active Nodal Task Seeking (ANTS) concept is used. Each computing node is an Active Nodal Task-Seeker (ANT) which, when it becomes idle, finds an appropriate, needed task for itself. For convenience, we use the above four phases as guideposts to describe the behavior of an ANTS sys-

1. Task definition: The task characteristics are defined at compilation time. Thus, the ANTS approach does place a significant responsibility on the programmer and compiler to decompose the real-time work into a sequence of short-running task segments. The task segments must be short-running in order that processors will become available often enough to satisfy changing real-time priorities such as interrupts associated with available new information. For many applications, such as communication, active sonar, radar, and space exploration, this decomposition is infrequently needed after the initial design of the system. It is essentially a one-time initial investment of effort.

2. Task assignment: The predecessor/successor relationships of each task and its current priority are the only guide lines for choosing the task segment which is to be executed. If all the predecessor conditions of a task segment are satisfied, and if there are no higher priority tasks to be carried out, one of the ANT nodes becoming available will find and execute this task segment after it begins to seek out needed work.

3. Task allocation: An ANT node finds one of the needed executable task segments according to the priorities attached to such tasks. An ANT node decides that a task is executable if it finds from a task table that the predecessor conditions of same task segment are satisfied.

4. Task migration: Since an ANT node seeks out a task if and only if that ANT is idle, there is always a strong tendency to distribute needed work and there is no load balancing or load sharing problem. If an ANT node fails while it is executing a task and the failure is identified (say by a task timer interrupt because the task took too

long), the same task is made available for the next ANT node. In other words, there is a simple and straightforward task migration due to the resulting reconfiguration of the system, as the failed node is assigned to a separate system partition for testing or replacement.

5.2 Comparisons

Table 1 summarizes the difference between the ANTS concept and conventional approaches. The above statements are independent of the details of the underlying architecture. If a specific architecture is considered, such as multiple bus, hypercube, *etc.*, other considerations must be addressed. For example, the communication costs between different sets of computing nodes vary significantly from a hypercube system to a local area network (LAN) connected set of processors. In that case, the seeking methods of the ANT nodes must be matched to the type of interprocessor communication (IPC) [25]. Nonetheless, the ANTS concept guarantees that almost no idle computing node exists in the system unless there are no more tasks listed in the task table or unless there are long communication delays. Dynamic task scheduling, such as rough grammar approach [26], can achieve about 0.6 computing node busyness, which means about 40% idle nodes on average. We expect a much higher rate of computing node busyness using the ANTS concept.

Task assignment, task allocation and task migration are all computationally complex. For instance, task (module) assignment problem is NP-hard in general [23]. Optimal task allocation or scheduling is NP-complete [26]. Even though polynomial time heuristic algorithms exist for these functions, their computational complexity is still an overhead to the system performance. Using the ANTS approach, there is no such computational overhead.

Communication overhead still exists using ANTS, but, such communication costs exist in any distributed system. There appears to be little, if any, extra communication required by ANTS. For example, in ANTS approach, there is no need for broadcasting. Conventional approaches usually rely on broadcasting or similar mechanism to maintain awareness of the current system status [27]. However, broadcasting is the most costly form of communication in a distributed system. Approaches that avoid the need for broadcasting, such as the buddy set in [27], could reduce this communication cost but the result will not be optimal.

The gain in computing node busyness using ANTS will outweigh any slight inefficiency in communications.

We further note that task definition, task assignment, and task allocation are job oriented. They, when successfully applied, find the optimal, or near optimal, solution to distributively execute a given job. If the entire system is used to execute a single job, an optimal execution time is achieved. On the other hand, under the same conditions, an ANTS system may not complete this one job in the optimal time. However, a typical system is designed to handle more than one job at a time. When several jobs need to be executed, the conventional approaches can only guarantee the optimality of each job but not the overall system performance. In other words, local optimal is reached but not the global optimal. An evidence of this shortcoming is the existence of task migration or load balancing and load sharing problems. A further evidence is that a typical distributed processing system has a high computing node idle rate while the task table is not empty. At this end, ANTS system outperforms the conventional approach in achieving the global optimal. Because the ANTS concept is similar to a greedy algorithm in that the first priority of an ANT node is to keep busy. When the computing node idle rate is low, a higher performance may be achieved.

By changing the way the tasks are distributed, the ANTS concept completely changes the way we deal with the problem of making distributed systems efficient and effective. In conventional approaches, a distributed system requires a different algorithm in each of the four phases which are listed above. Although extra one-time work is required from the programmer and compiler, a distributed system using ANTS needs only one type of algorithm to achieve the efficiency. Besides, the ANTS concept also provides easy implementation of fault-tolerant techniques to efficiently achieve a desired level of dependability. In summary, the ANTS concept appears to be a useful concept for achieving both high-performance and ultra-dependability for a real-time distributed system.

6 Conclusions

We have presented in this paper the dependability analysis of the ANTS ultra-dependable multicomputer system and we have argued that high-performance and efficient use of available processing resources can also be attained. We have shown that the concept of run-time

partitioning for diagnostic program checking and the re-enlistment greatly enhance the mean time to failure of the system. The goal of more than 20 years of MTTF could be easily achieved with computing node failure rate of 5×10^{-4}, or MTTF=2,000 hours. This means that, in ANTS, there is no need for ultra-dependable computing nodes. An inexpensive implementation is feasible. Of course, for harsh environment such as military applications, where failure rate is high, ultra-dependable components are still required.

In [5], the Weibull distribution is recommended for its capability to include the aging effect in failure rate modeling. Even though we use constant failure rate (exponential distribution) in this analysis, we show that the ultra-dependability of ANTS was not dependent on the low failure rate of components. Of course, verification of these analysis results requires further investigation involved fault/error injection experiments. We are currently working on an emulation system of ANTS with fault/error injections from both hardware and software sources. Also, further work is needed to quantify our performance analysis.

Acknowledgement

The authors thank the editor and the anonymous reviewers for their constructive comments. We also thank Mr. Patrick Dominic-Savio for his help in preparation of this paper.

References

[1] K. Hwang and F. A. Briggs, *Computer Architecture and Parallel Processing.* McGraw-Hill Inc., 1984.

[2] H. Stone, *High-Performance Computer Architecture.* Addison-Wesley, 1987.

[3] J. L. Hennessy and D. A. Patterson, *Computer Architecture: A Quantitative Approach.* Morgan Kaufmann Publishers Inc., Palo Alto, CA, 1990.

[4] AT&T, "Safeguard supplement," *The Bell System Tech. J.*, 1975.

[5] D. P. Siewiorek and R. S. Swarz, *Reliable Computer Systems: Design and Evaluation.* 2nd edition, Digital Press, 1992.

[6] W. N. Toy, "Fault-tolerant design of local ESS processors," *Proc. IEEE*, pp. 1126–1145, October 1978.

[7] A. L. Hopkins, Jr., T. B. Smith III, and J. H. Lala, "FTMP - A highly reliable fault-tolerant multiprocessor for aircraft," *Proc. IEEE*, pp. 1221–1239, October 1978.

[8] J. H. Wensley, L. Lamport, J. Goldberg, M. W. Green, K. N. Levitt, P. M. Melliarsmith, R. E. Shostak, and C. B. Weinstock, "SIFT: Design and analysis of a fault-tolerant computer for aircraft control," *Proc. IEEE*, pp. 1240–1255, October 1978.

[9] M. J. Iacoponi and D. K. Vail, "The fault tolerance approach of the advanced architecture on-board processor," in *Proc. 19th Int'l Symp. Fault-Tolerant Comput.*, pp. 6–12, June 1989.

[10] M. J. Iacoponi and S. F. McDonald, "Distributed reconfiguration and recovery in the advanced architecture on-board processor," in *Proc. 21st Int'l Symp. Fault-Tolerant Comput.*, pp. 436–443, June 1991.

[11] F. Cristian, B. Dancey, and J. Dehn, "Fault-tolerant in the advanced automation system," in *Proc. 20th Int'l Symp. Fault-Tolerant Computing*, pp. 6–17, June 1990.

[12] T. R. Dilenno, D. A. Yaskin, and J. H. Barton, "Fault-tolerant testing in the advanced automation system," in *Proc. 21st Int'l Symp. Fault-Tolerant Comput.*, pp. 18–25, June 1991.

[13] T. Takano, T. Yamada, K. Shutoh, and N. Kanekawa, "Fault-tolerant experiments of the "Hiten" onboard space computer," in *Proc. 10th Int'l Symp. Fault-Tolerant Comput.*, pp. 26–33, June 1991.

[14] M. Chereque, D. Powell, P. Reynier, J.-L. Richier, and J. Voiron, "Active replication in Delta-4," in *Proc. 22nd Int'l. Symp. Fault-Tolerant Comput.*, pp. 28–37, July 1992.

[15] E. N. Elnozahy and W. Zwaenepoel, "Replicated distributed processes in Manetho," in *Proc. 22nd Int'l Symp. Fault-Tolerant Computing*, pp. 18–27, July 1992.

[16] N. G. Shivaratri, P. Krueger, and M. Singhal, "Load distributing for locally distributed systems," *IEEE Computer*, pp. 33–44, December 1992.

[17] L. Cheng and A. Avizienis, "N-version programming: A fault tolerance approach to reliability of software operation," in *Proc. 8th Int'l. Symp. Fault-Tolerant Comput.*, pp. 3–9, June 1978.

[18] J. R. Sklaroff, "Redundancy management technique for space shuttle computers," *IBM I. Res. Develop.*, pp. 20–27, January 1976.

[19] Y. Savaria, N. C. Rumin, J. F. Hayes, and V. K. Agarwal, "Soft-error filtering: A solution to the reliability problem of future VLSI digital circuits," *Proc. IEEE*, pp. 669–683, May 1984.

[20] K. S. Trivedi, *Probability & Statistics with reliability, Queuing, and Computer Science Applications.* Prentice-Hall, Inc., Englewood Cliffs, NJ, 1982.

[21] J. E. Angus, "On computing MTBF for a k-out-of-n:G repairable system," *IEEE Trans. Reliability*, pp. 312–313, August 1988.

[22] V. M. Lo, "Heuristic algorithms for task assignment in distributed systems," *IEEE Trans. Comput.*, pp. 1384–1397, November 1988.

[23] D. Fernandez-Baca, "Allocating modules to processors in a distributed system," *IEEE Tran. Software Eng.*, pp. 1427–1436, November 1989.

[24] P. D. Hoang and J. M. Rabaey, "Scheduling of DSP programs onto multiprocessors for maximum throughput," *IEEE Trans. Signal Proc.*, pp. 2225–2235, June 1993.

[25] W. W. Chu, L. J. Holloway, M. T. Lan, and K. Efe, "Task allocation in distributed data processing," *IEEE Computer*, pp. 57–69, November 1980.

[26] Z. M. Wojcik and B. E. Wojcik, "Rough grammar for efficient and fault-tolerant computing on a distributed system," *IEEE Trans. Software Eng.*, pp. 652–668, July 1991.

[27] K. G. Shin and Y. C. Chang, "Load sharing in distributed real-time systems with state-change roadcasts," *IEEE Trans. Comput.*, pp. 1124–1142, August 1989.

Table 1. A comparative view between the ANTS concept and conventional approaches.

Phase	Conventional Approaches	ANTS
Task Definition *Problem*: Partition a job into a set of tasks to be executed distributively.	Determine the data flow dependancy; estimate run time of tasks; etc.	Same as that in conventional approaches. Except that the running time of each task segment is short and within pre-defined limits.
Task Assignment *Problem*: Assigning m tasks to p processors, such that a high processor utilization is ensured and that the shortest execution time is achieved.	1. NP-hard problem. 2. Polynomial time heuristic algorithm is used to find a near optimal solution.	No pre-run-time assignment is neccessary, because of the separation of each job into short running task segments by the programmer and compiler.
Task Allocation *Problem*: Allocating tasks to processors according to the current system status, *e.g.*, cost of inter-processor communication (IPC), number of available processors, etc.	1. NP-complete problem. 2. Polynomial time heuristic algorithm is used to find a near optimal solution. 3. Require constant update of system status by means of broadcasting or similar mechanism.	An ANT node finds one of the needed executable tasks according to the prorities attached to such tasks. An ANT node decides that a task is executable if it finds from a task table that the predecessor conditions of that task are satisfied.
Task Migration (I) *Problem*: Migrating tasks to different nodes due to the changes in the system status to maintain load balancing.	Require constant update of system status by means of broadcasting or similar mechanism.	System load is always balanced since an ANT node seeks out a task if and only if it is idle.
Task Migration (II) *Problem*: Migrating tasks to fault-free nodes after system reconfiguration.	Task allocation may be neccessary after a system reconfiguration.	The task being executed by the failed node, once the failure is identified, is made available for the next ANT node.

Remarks:

1. The ANTS concept uniformly handles all four phases. Conventional approaches solve problems in respective phases.
2. The ANTS approach needs little or no computational time and resources.

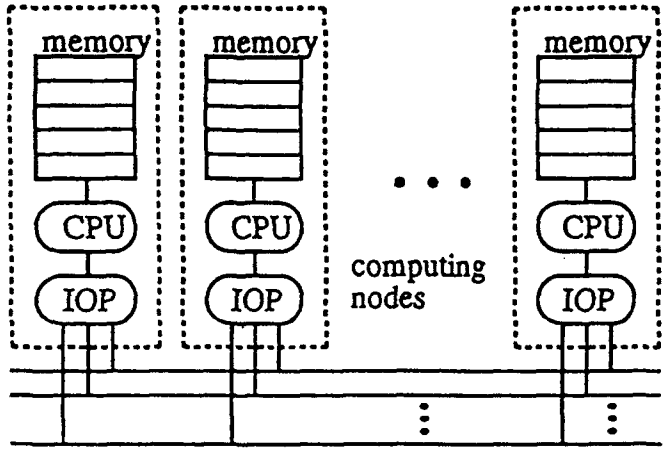


Figure 1: Organization of an implementation of ANTS.

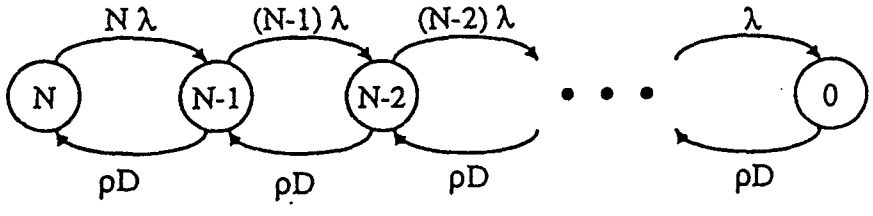


Figure 2: Birth and Death Markov Model of ANTS Subsystem.

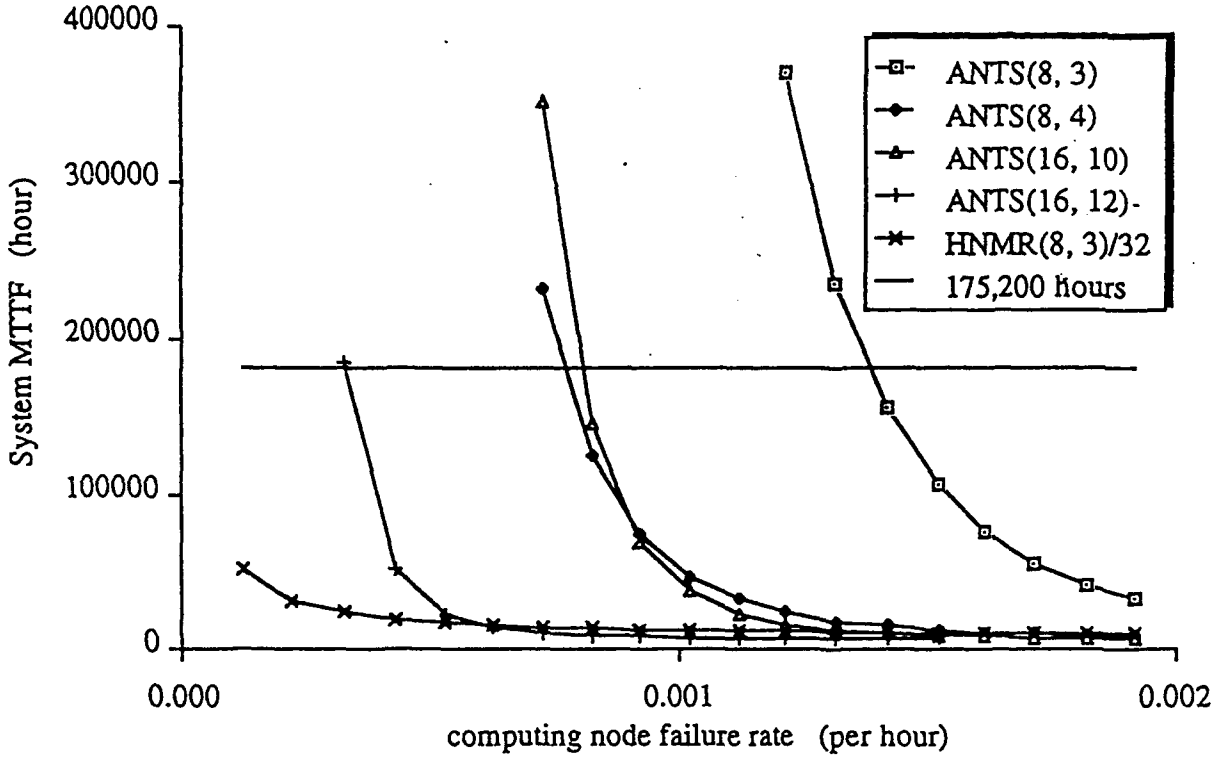


Figure 3: MTTF of system with KB=1, NB=3, λ_b=1×10^{-5} per hour, ρ=0.1 per hour and D=0.8. The HNMR(8, 3)/32 denotes a Hybrid N-Modular Redundancy system with 32 standby spares.

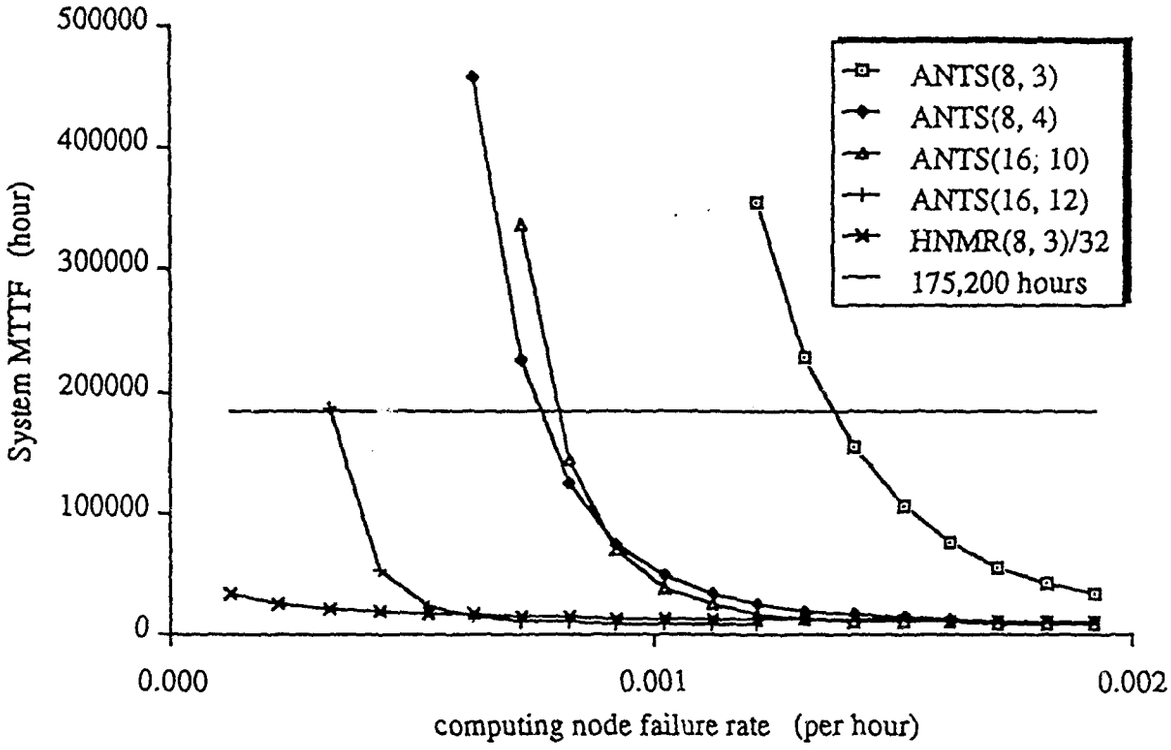


Figure 4: MTTF of system with KB=1, NB=3, λ_b=1×10^{-4} per hour, ρ=0.1 per hour and D=0.8. The HNMR(8, 3)/32 denotes a Hybrid N-Modular Redundancy system with 32 standby spares.

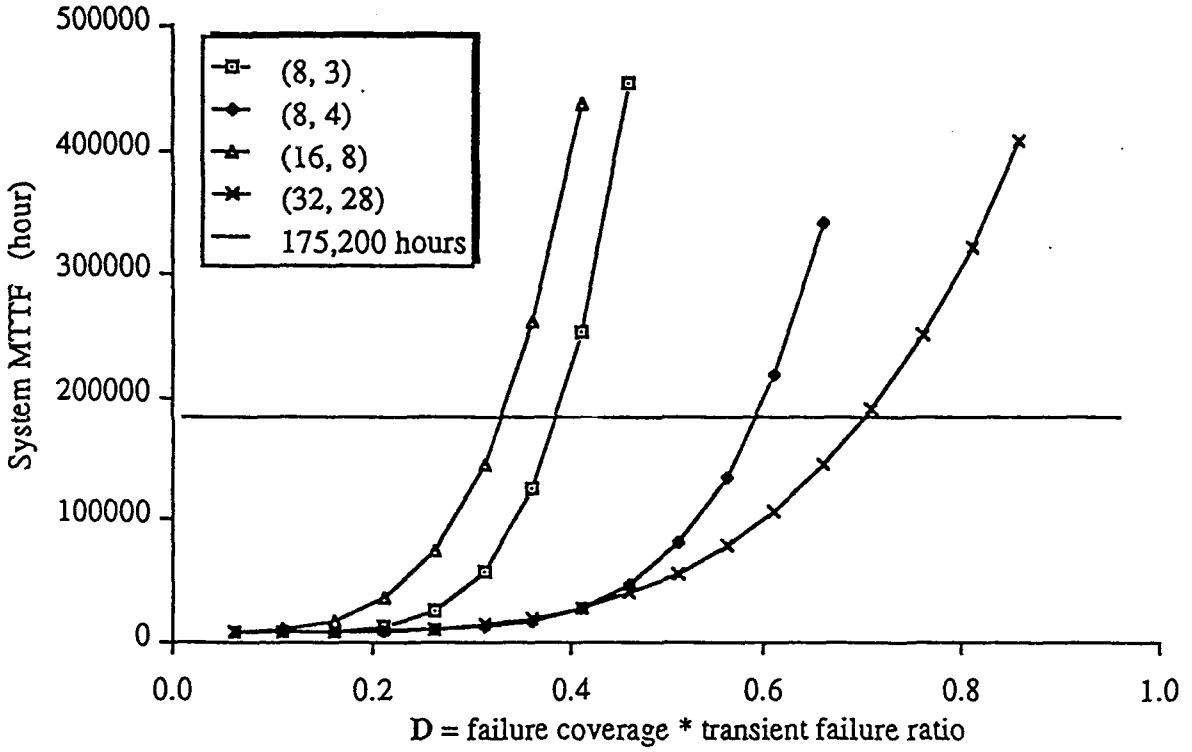


Figure 5: MTTF of ANTS system with KB=1, NB=3, λ_b=1×10^{-4} per hour, $\lambda_p = 5 \times 10^{-4}$ per hour and ρ =0.1 per hour.

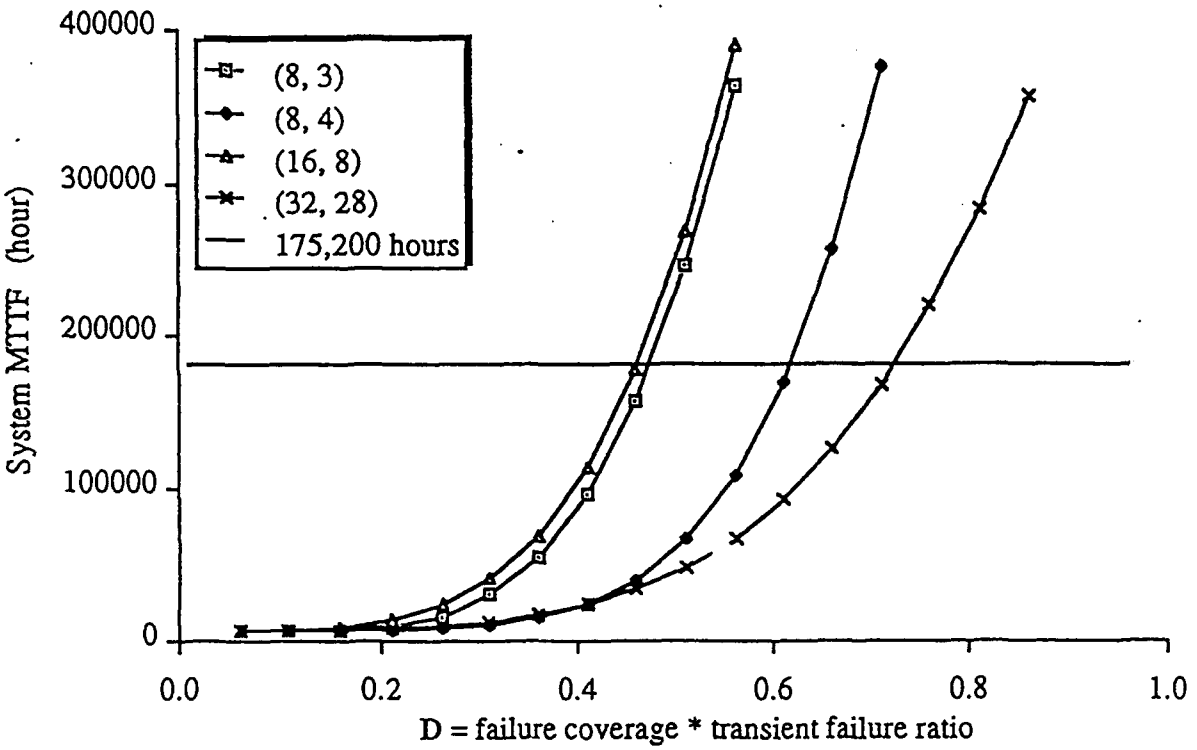


Figure 6: MTTF of ANTS system with KB=2, NB=4, λ_b=1×10^{-4} per hour, $\lambda_p = 5 \times 10^{-4}$ per hour and ρ =0.1 per hour.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	July 1994	Semi annual 1/15/94 - 7/15/94

4. TITLE AND SUBTITLE

Active Nodal Task Seeking for High-Performance, Ultra-Dependable Computing

5. FUNDING NUMBERS

G:

N00014-94-1-0479

6. AUTHOR(S)

Jien-Chung Lo, Donald W. Tufts and James W. Cooley

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Department of Electrical & Computer Engineering
University of Rhode Island
Kingston, RI 02881-0805

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

Office of Naval Research
Ballston Tower One
800 North Quincy Street
Arlington, Virginia 22217-5660

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

This paper is scheduled for publication in an upcoming issue of IEEE Transactions on Aerospace & Electronic Systems (Vol. 31, No.4, October 1995).

12a. DISTRIBUTION/AVAILABILITY STATEMENT

12b. DISTRIBUTION CODE

13. ABSTRACT (Maximum 200 words)

The Active Nodal Task Seeking(ANTS) approach to the design of multicomputer systems is named for its basic component: an Active Nodal Task-Seeker(ANT). In this system, there is no load balancing or load sharing, instead, each ANT computing node is actively finding out how it can contribute to the execution of the needed tasks. A run-time partition is established such that some of the ANT computing nodes are under exhaustive diagnosis at any given time. An ANTs multicomputer system can achieve a mean time to failure of more than 20 years with just 8 computing nodes and 3 buses, while the minimum requirements are 3 computing nodes and 1 bus, and with a worst case computing node failure rate of 5 x 0.0001 per hour. This work has been motivated by the need to develop high-performance multicomputer systems for radar, active and passive sonar, and electronic warfare that can provide ultra-dependable performance for more than 20 years without field repairs. We argue that high performance is also an attribute of an ANTS computing system, because the overhead of dynamic task scheduling is reduced and because efficient use is made of the available processing resources.

14. SUBJECT TERMS

Dependable distributed system, fault-tolerant computing, active mode operating system, mean time to failure, high-performance

15. NUMBER OF PAGES

28

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UL